08 april 2026

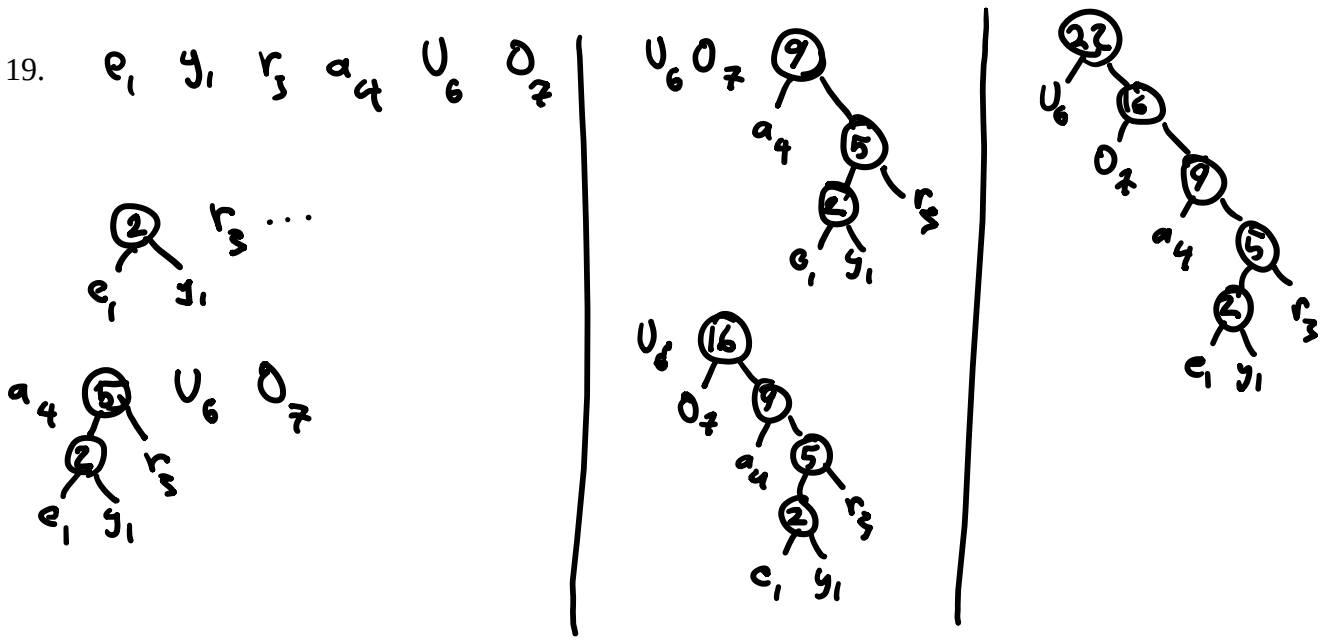
Planlagt: 09:00-13:00

Eksamensnr: 292

Skriv IKKE her

Start på side 2

19.


e1 y1 r3 a4 U6 O7
2
e1 y1
r3 ...
a4 5 U6 O7
2 r3
e1 y1
U6 O7 9
a4 5
2 r3
e1 y1
U6 16
O7 9
a4 5
2 r3
e1 y1
22
U6 16
O7 9
a4 5
2 r3
e1 y1

20.

a)

We are looking for a sum of some numers from respectrive array so that the sum appears in all 3 arrays. The sum includes all nrs up to index but index can vary.

All sums for

A: 1,3,6,10

B: 3,5,8,12

C: 1,2,3,7

Sums that appear in all:

1,3, where as 3 is the largest. ANS: 3

b)

i) a variable largest_prefix_sum, and a dictionary D-

.

ii) we loop through 1 to n, and for each iiteration get elements a[i], b[i]. c[i]

iii) each itteration, calcualte a cumulative sum for each list as c = c + element[i],

iiii)

We attempt an insert each cumulative element into a dictionary key=element size, and element = a,b,c depending on in which list the element was observed.

iiiii) Whenever we insert we check if the elements are [a,b,c] if so we check if element is larger than largest_prefix, if so we update largest prefix.

we now have 3 cumulative lists we need to effectively check for elements that appear in all of them.

The first loop is 1,n hence O(n), so is the second loop. Hence the solution is T(n) = O(n)

c)

We want to show that our algo reutrns the largest prefix sum observed upto i=n in all 3 lists.

Invariant: . each itteration i largest_prefix_sum contains largest prefix sum observed that are in all arrays from 1-i.

B.s

We initialise empty lists, with zero itterations we have 0 prefix sum holds.

I.h

we assume it holds for i and we want to show that the algo correctly updates going to i+1

I.s

It holds that dictionary contains one key for all observed sums up to i, and that it is recorded which lists it was observed in.

We only update the max prefix sum variable if i) it is observed in all 3 lists, ii) its larger than the current prefix sum. It therefore contains the largest prefix sum observed upto i each itteration. The invaraiint holds for the update.

Therefore it holds that when the loop is doen we have the correct max prefix sum.

21.

a)

Here we choose what direction to check depending on mid element,

First mid = 5, larger than 1, hence we check 2,4 mid 2 larger than one, now p=q then again we check if mid=2 larger than 1, which again is false so now p neq q is false and loop exits and we reutrn false.

ANS: [2,4,5.3,1], x=1

b)

Assume bianry serach and reverser binary serach runs in O(logn) we have:

T(n) = has tow cases

1. Amid< Amid+1
   O(lgn) + T(n/2)
2. else
   O(lgn) + T(n/2)

Since each recursive call only one case can be true and both have the same time complexity we can simplify as:

T(n) = O(lgn) + T(n/2)

With master theorem w eget a=1, b=2, recursive part is n**lg2(1), 2**x=1? x=0 -> O(n**n)=O(1) hence the work each level O(lgn) dominates. The runtime is O(lgn)

EDIT Since we dont return false even if we directly ifnd element we actually run it more but i am running out of time just saw this, we have to wait for BTS2 to return so this runs at most n times so i think it should be O(nlogn) runtime. We do

c)

If strictly bitonic, we have two cases for possible mid elements, one where the next element is larger or where it is not. Depending on if we are in the first or second part of the bitonic list, were element size is increaseing (or decreasing in the second part).

We know binary serach assumes elements are sorted in non decreasing order, and hence reverse binary serach assumes non increasing order. Non decreasing order holds true for the first part of a bitonic list and non increasing holds true for the second part. Hence in essence binary serach will run correctly for the first part and reverse binary serach will run corretly for the second part.

Since weu se the if statment A[mid] < A[mid] +1 we correctly identify which part of the list we are looking at.

If we are looking for an element x in the first part where element size is increasing, we run binary serach, which will find the element in a list of increasing order. This will correecly find our element.

If x is in the second decreasing part of the list. The if statement Amid < Amid+1 is false and we fall back to the else statement. Here we do the same thing as in the first case but run reverse binary serach instead for that section of the list. m+1 to q. This is equivalent to running regular binary serach for an increasing list on the same list range. Hence this also correctly returns true if x is in that range.

1 Asymptotiske grænser

Nedenfor er angivet et antal udtryk for køretid. Hvilken køretid er asymptotisk langsomst voksende? [...]

5 log(4^n)

2 Rekursionsligninger

Antag at n er en potens af 3, og at T(1)=1. Betragt rekursionsligningen [...]

2 T(n) = Theta(n^log_3(5))

3 Del og hersk

En rekursiv algoritme SortAndRule med input af størrelse n (en potens af 2) består af tre dele: [...]

6 T(n) = 4 T(n/2) + Theta(n log n)

4 Løkkeinvarianter

Betragt følgende algoritme der som input tager et array A[1:n] og et naturligt tal n. [...]

4 Efter hver iteration gælder at m > =min{A[1],…,A[i-1]}

2 Ved starten af hver iteration gælder at m = max{A[1],…,A[i-1]}

5 Køretidsanalyse

Lad RB-Tree-Create() betegne en procedure der skaber et rød-sort søgetræ (uden nøgler) som beskrevet i CLRS sektion 13. [...]

5 Theta(n log n + m log n)

6 Amortiseret analyse

Betragt dynamiske tabeller som beskrevet i CLRS sektion 16.4.1 (uden sletninger) men med følgende variant af Table-Insert(T,x) hvor tabelstørrelsen udvides med en faktor k, hvor k> = 2 er en heltalsparameter: [...]

4 Phi(T) = k (T.num - T.size/k)

7 LSM-træer

Betragt et simpelt LSM-træ, som beskrevet i noten om LSM-træer, hvor alle sorterede lister har længde der er en potens af 2, og hvor der for hver størrelse 2^i findes højst én (sorteret) liste. Hvad kan man sige om antal binære søgninger (i ikke-tomme lister) der skal udføres for at søge efter en nøgle i et LSM-træ med n > 15 elementer? [...]

1 Aldrig større end lglg(n+1)

8 Binære søgetræer

Betragt et tomt binært søgetræ (BST), uden balancering som beskrevet i CLRS kapitel 12. [...]

1 Resultatet er 1,3,4,5,7,8,10,12

9 Rød-sorte søgetræer

Betragt dette rød-sorte binære søgetræ (eng. red-black binary search tree), hvor bladene er udeladt på tegningen: [...]

3 42 indsættes som venstre barn til knude 58 og farves rød

10 Bellman-Ford

Betragt følgende orienterede, vægtede graf G: [...]

1 (d[s],d[a],d[b],d[c]) = (0,4,2,3)

11 Gør algoritmen færdig

Betragt følgende algoritme (med en uspecificeret del), der som input tager et array af heltal A[1:n] og positive heltal n, k. [...]

4 max{B[i-1,j],B[i-1,j-1]+A[i]}

12 Mindste udspændende træ

Betragt mindste udspændende træ (eng. minimum spanning tree) problemet på denne graf, med en uspecificeret vægt x: [...]

1 x=1

2 x=3

3 x=5

13 Disjunkte mængder

Vi betragter trærepræsentationer af disjunkte mængder (eng. disjoint sets), som beskrevet i CLRS sektion 19.3, ved brug af weighted-union heuristikken uden brug af stiforkortning. [...]

4 Tiden er O(log m) for alle x

14 Korteste veje

Betragt følgende vægtede graf G=(V,E): [...]

3 5 kanter

15 Valg af algoritme

Vi ønsker at finde en sti gennem et 2-dimensionelt gitter fra en startposition S til en slutposition T, der har en så lav omkostning som muligt. [...]

2 Dijkstra's algoritme

16 Nærmeste par af punkter

I CLRS digitalt kapitel 33, sektion 4, præsenteres en del-og-hersk algoritme med køretid O(n log n) til at finde nærmeste par af punkter i en mængde af 2-dimensionelle punkter. [...]

3 For alle input kører algoritmen i tid Theta(n log n)

17 Længste fælles delsekvens

Hvor lang er længste fælles delsekvens af strengene AABBCDAAB og ABDAACBB? [...]

5 Længde 5

18 Arbejde og spand

Betragt følgende parallelle algoritme, der finder maksimum i et array A[1:n] ved divide-and-conquer. [...]

3 Arbejde T_1=Theta(n), spand T_infinity=Theta(log n)